

# The Need for Domain-Specific Solutions

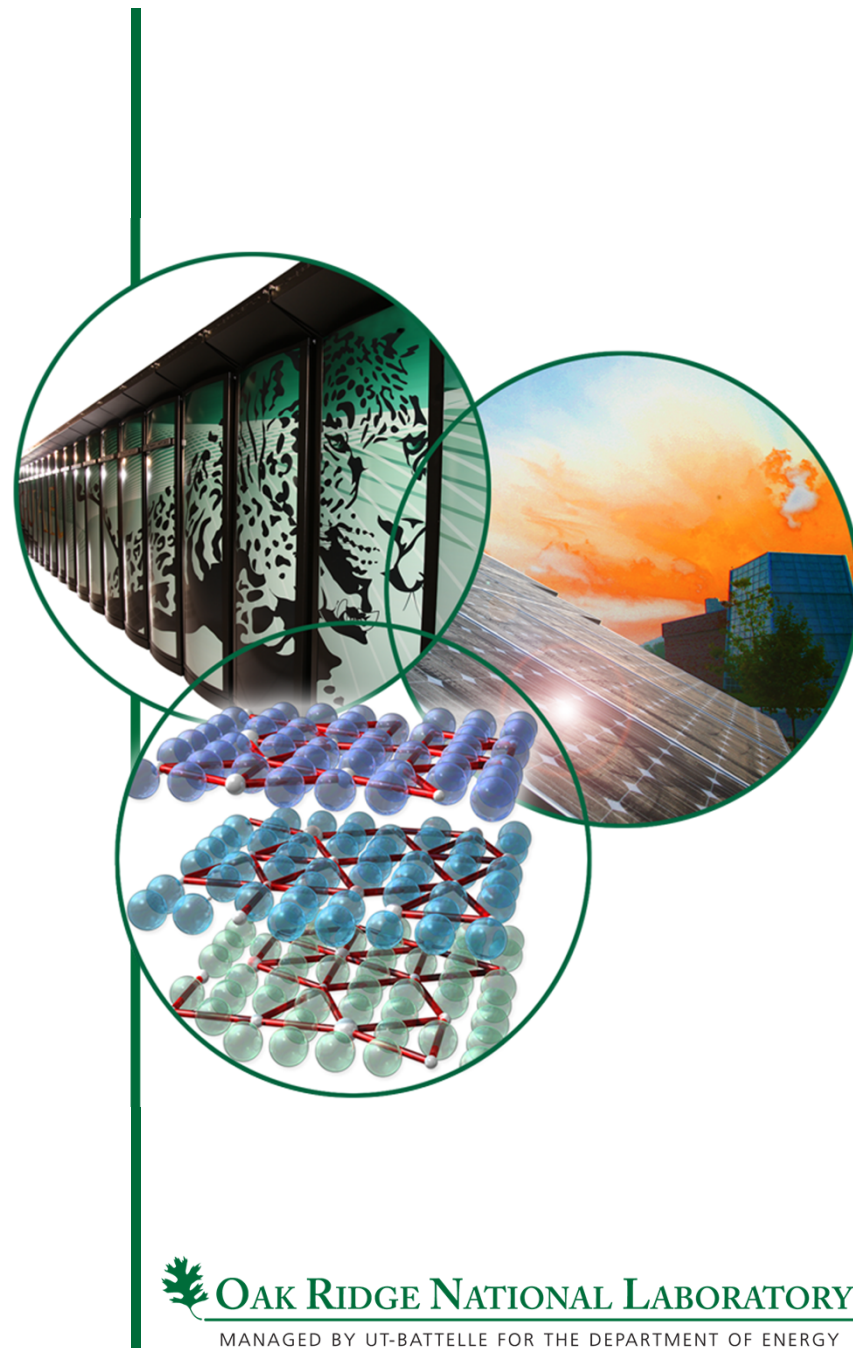
David E. Bernholdt

Oak Ridge National Laboratory  
*bernholdtde@ornl.gov*



12-15 March 2012

SOS 16



 **OAK RIDGE NATIONAL LABORATORY**  
MANAGED BY UT-BATTELLE FOR THE DEPARTMENT OF ENERGY

# The Costs of Exascale Computing won't be Limited to Hardware

- The computational science & engineering community relies extensively on large, long-lived codes
  - $O(100k)$  lines typical, some  $O(1M)$  lines or more
  - Lifespans often measured in decades
- Taking full advantage of exascale systems will require significant changes, rewrites (1.5x)
  - Exposing and managing parallelism, large node parallelism, multilevel parallelism, accelerators (or not)
  - Exposing and managing locality & data movement
  - Energy and power constraints
  - Limited memory, limited I/O (bandwidth & capacity)
  - Resilience concerns exposed to programmer
  - FLOPS free/data movement expensive, new algorithms?

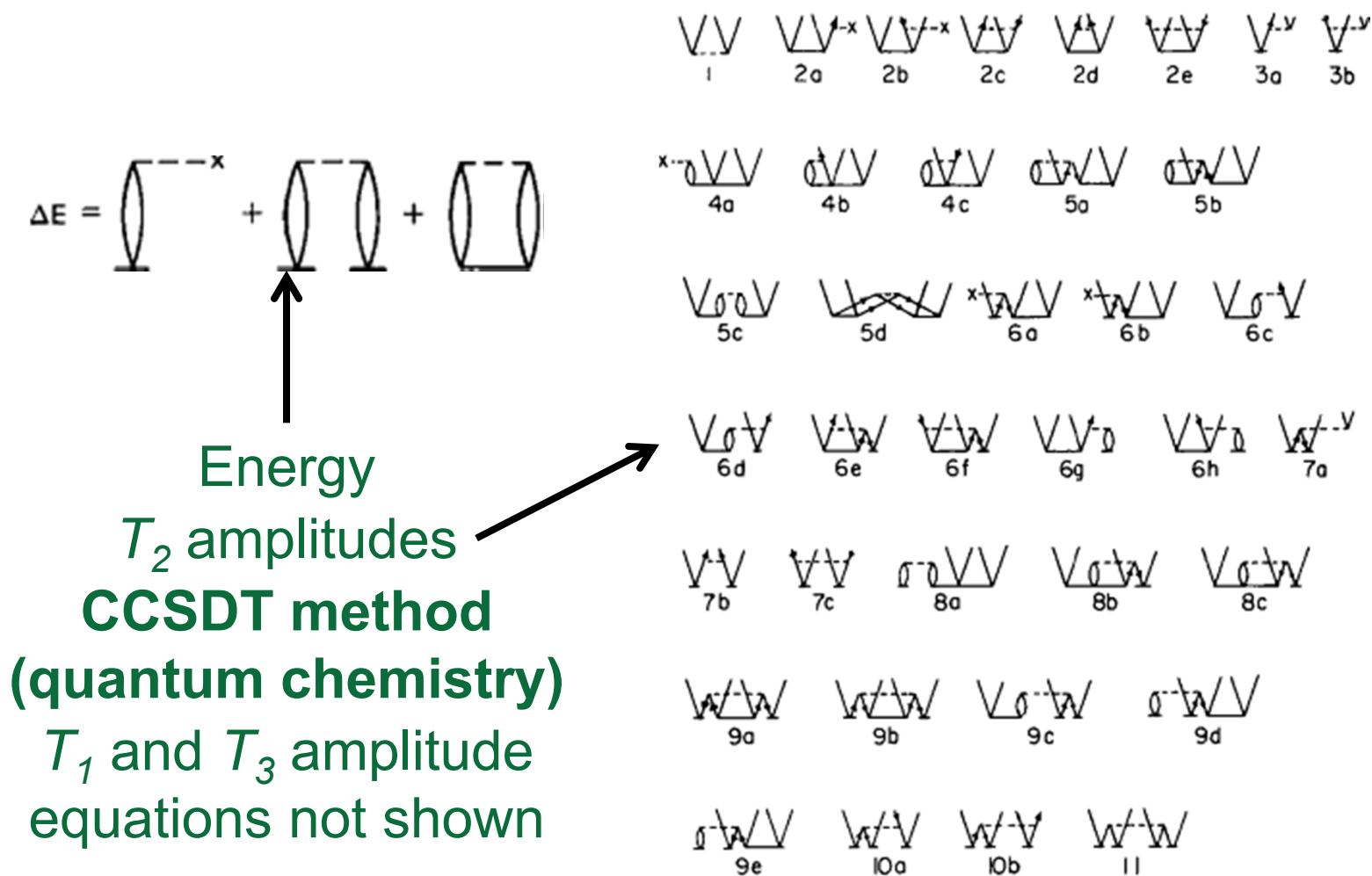
# Reducing the Costs of Application Software for Exascale

- The number of lines of code a programmer can write in a fixed period of time is the same independent of the language used (Corbato's Law)
  - Productivity and reliability depend on the length of the code, not the language used
- Create a programming environment that better matches the characteristics of exascale-era hardware
  - Reduce the cost of mapping the code onto the hardware
  - Today's programming environments are based on hardware 30+ years old with 20 year old ideas bolted on
- Create a programming environment that better matches the characteristics of the scientific problem being solved
  - Reduce the cost of mapping the equations into code

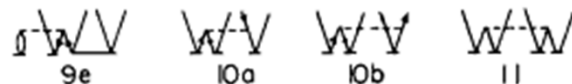
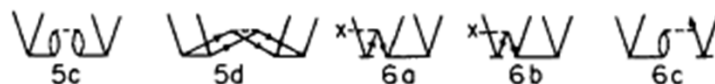
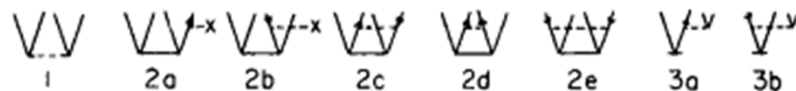
# Domain-Specific Languages (DSLs)

- Programming language dedicated to a particular problem domain, a particular problem representation technique, and/or a particular solution technique
  - Libraries may be used in a similar sense
  - Example domains (from WOLFHPC11): PDEs, relativistic spacetime, preconditioned iterative solvers, dense linear algebra, quantum chemistry, stencil computations, OpenCL
- Benefits for scientist-programmers...
  - Express computations at a higher level of abstraction – more compact code
  - Closer to the way they think about/publish problem
  - Focused (constrained), natural environment makes errors less likely, better error messages make debugging easier
  - Let compiler worry about how to implement most efficiently for target platform

# Sometimes the Equations Don't Even Look Like Equations!



# CCSD $T_2$ Amplitude Equation



# CCSD $T_2$ Amplitude Equation

$$\begin{aligned}
0 = & \langle ab || ij \rangle + \sum_c (f_{bc} t_{ij}^{ac} - f_{ac} t_{ij}^{bc}) - \sum_k (f_{kj} t_{ik}^{ab} - f_{ki} t_{jk}^{ab}) + \\
& \frac{1}{2} \sum_{kl} \langle kl || ij \rangle t_{kl}^{ab} + \frac{1}{2} \sum_{cd} \langle ab || cd \rangle t_{ij}^{cd} + P(ij) P(ab) \sum_{kc} \langle kb || cj \rangle t_{ik}^{ac} + \\
& P(ij) \sum_c \langle ab || cj \rangle t_i^c - P(ab) \sum_k \langle kb || ij \rangle t_k^a + \\
& \frac{1}{2} P(ij) P(ab) \sum_{klcd} \langle kl || cd \rangle t_{ik}^{ac} t_{lj}^{db} + \frac{1}{4} \sum_{klcd} \langle kl || cd \rangle t_{ij}^{cd} t_{kl}^{ab} - \\
& P(ab) \frac{1}{2} \sum_{klcd} \langle kl || cd \rangle t_{ij}^{ac} t_{kl}^{bd} - P(ij) \frac{1}{2} \sum_{klcd} \langle kl || cd \rangle t_{ik}^{ab} t_{jl}^{cd} + \\
& P(ab) \frac{1}{2} \sum_{kl} \langle kl || ij \rangle t_k^a t_l^b + P(ij) \frac{1}{2} \sum_{cd} \langle ab || cd \rangle t_i^c t_j^d - P(ij) P(ab) \sum_{kc} \langle kb || ic \rangle t_k^a t_j^c + \\
& P(ab) \sum_{kc} f_{kc} t_k^a t_{ij}^{bc} + P(ij) \sum_{kc} f_{kc} t_i^c t_{jk}^{ab} - \\
& P(ij) \sum_{klc} \langle kl || ci \rangle t_k^c t_{lj}^{ab} + P(ab) \sum_{kcd} \langle ka || cd \rangle t_k^c t_{ij}^{db} + \\
& P(ij) P(ab) \sum_{kcd} \langle ak || dc \rangle t_i^d t_{jk}^{bc} + P(ij) P(ab) \sum_{klc} \langle kl || ic \rangle t_i^a t_{jk}^{bc} + \\
& P(ij) \frac{1}{2} \sum_{klc} \langle kl || cj \rangle t_i^c t_{kl}^{ab} - P(ab) \frac{1}{2} \sum_{kcd} \langle kb || cd \rangle t_k^a t_{ij}^{cd} - \\
& P(ij) P(ab) \frac{1}{2} \sum_{kcd} \langle kb || cd \rangle t_i^c t_k^d t_j^a + P(ij) P(ab) \frac{1}{2} \sum_{klc} \langle kl || cj \rangle t_i^c t_k^a t_l^b - \\
& P(ij) \sum_{klcd} \langle kl || cd \rangle t_k^c t_i^d t_{lj}^{ab} - P(ab) \sum_{klcd} \langle kl || cd \rangle t_k^c t_l^a t_{ij}^{db} + \\
& P(ij) \frac{1}{4} \sum_{klcd} \langle kl || cd \rangle t_i^c t_j^d t_{kl}^{ab} + P(ab) \frac{1}{4} \sum_{klcd} \langle kl || cd \rangle t_k^a t_l^b t_{ij}^{cd} + \\
& P(ij) P(ab) \sum_{klcd} \langle kl || cd \rangle t_i^c t_l^b t_{kj}^{ad} + P(ij) P(ab) \frac{1}{4} \sum_{klcd} \langle kl || cd \rangle t_i^c t_k^a t_j^d t_l^b.
\end{aligned}$$

# CCSD $T_2$ Amplitude Equation

$$\begin{aligned} \text{hbar}[a,b,i,j] = & \text{sum}[f[b,c]*t[i,j,a,c],\{c\}] - \text{sum}[f[k,c]*t[k,b]*t[i,j,a,c] \\ & \text{sum}[f[k,j]*t[i,k,a,b],\{k\}] - \text{sum}[f[k,c]*t[j,c]*t[i,k,a,b],\{k,c\}] - \text{sum} \\ & + \text{sum}[t[i,j,c,d]*v[a,b,c,d],\{c,d\}] + \text{sum}[t[j,c]*v[a,b,i,c],\{c\}] - \text{sum} \\ & \text{sum}[t[k,d]*t[i,j,c,b]*v[k,a,c,d],\{k,c,d\}] - \text{sum}[t[i,c]*t[j,k,b,d]*v[l \\ & \text{sum}[t[j,k,c,b]*v[k,a,c,i],\{k,c\}] - \text{sum}[t[i,c]*t[j,d]*t[k,b]*v[k,a,d,c] \\ & \text{sum}[t[k,b]*t[i,j,c,d]*v[k,a,d,c],\{k,c,d\}] - \text{sum}[t[j,d]*t[i,k,c,b]*v[l \\ & \text{sum}[t[i,c]*t[j,k,d,b]*v[k,a,d,c],\{k,c,d\}] - \text{sum}[t[j,k,b,c]*v[k,a,i,c] \\ & \text{sum}[t[i,c]*t[j,d]*t[k,a]*v[k,b,c,d],\{k,c,d\}] - \text{sum}[t[k,d]*t[i,j,a,c]* \\ & + 2*\text{sum}[t[j,d]*t[i,k,a,c]*v[k,b,c,d],\{k,c,d\}] - \text{sum}[t[j,d]*t[i,k,c,a] \\ & \text{sum}[t[i,c]*t[k,a]*v[k,b,c,j],\{k,c\}] + 2*\text{sum}[t[i,k,a,c]*v[k,b,c,j],\{ \\ & \text{sum}[t[j,d]*t[i,k,a,c]*v[k,b,d,c],\{k,c,d\}] - \text{sum}[t[j,c]*t[k,a]*v[k,b, \\ & + \text{sum}[t[i,c]*t[j,d]*t[k,a]*t[l,b]*v[k,l,c,d],\{k,l,c,d\}] - 2*\text{sum}[t[k,b] \\ & + \text{sum}[t[j,d]*t[l,b]*t[i,k,c,a]*v[k,l,c,d],\{k,l,c,d\}] - 2*\text{sum}[t[i,c]*t[l \\ & + \text{sum}[t[i,c]*t[l,b]*t[j,k,d,a]*v[k,l,c,d],\{k,l,c,d\}] + \text{sum}[t[i,k,c,d]* \\ & 2*\text{sum}[t[i,k,c,a]*t[j,l,b,d]*v[k,l,c,d],\{k,l,c,d\}] - 2*\text{sum}[t[i,k,a,b]* \\ & + \text{sum}[t[i,k,c,a]*t[j,l,d,b]*v[k,l,c,d],\{k,l,c,d\}] + \text{sum}[t[i,c]*t[j,d]*t \\ & 2*\text{sum}[t[i,j,c,b]*t[k,l,a,d]*v[k,l,c,d],\{k,l,c,d\}] - 2*\text{sum}[t[i,j,a,c]* \\ & + \text{sum}[t[i,c]*t[j,k,b,a]*v[k,l,c,i],\{k,l,c\}] - 2*\text{sum}[t[l,a]*t[j,k,b,c]*v \\ & 2*\text{sum}[t[k,c]*t[j,l,b,a]*v[k,l,c,i],\{k,l,c\}] + \text{sum}[t[k,a]*t[j,l,b,c]*v \\ & + \text{sum}[t[j,c]*t[l,k,a,b]*v[k,l,c,i],\{k,l,c\}] + \text{sum}[t[i,c]*t[k,a]*t[l,b]* \\ & 2*\text{sum}[t[l,b]*t[i,k,a,c]*v[k,l,c,j],\{k,l,c\}] + \text{sum}[t[l,b]*t[i,k,c,a]*v \\ & + \text{sum}[t[j,c]*t[l,d]*t[i,k,a,b]*v[k,l,d,c],\{k,l,c,d\}] + \text{sum}[t[j,d]*t[l,b] \\ & 2*\text{sum}[t[i,k,c,d]*t[j,l,b,a]*v[k,l,d,c],\{k,l,c,d\}] - 2*\text{sum}[t[i,k,a,c]* \\ & + \text{sum}[t[i,k,a,b]*t[j,l,c],\{k,l,c\}] - 2*\text{sum}[t[i,k,a,b]*t[j,l,c],\{k,l,c\}] \\ & + \text{sum}[t[k,a]*t[l,b]*v[k,l,c,i],\{k,l,c\}] - 2*\text{sum}[t[l,a]*t[j,k,b,c]*v \\ & + \text{sum}[t[k,a]*t[l,d]*t[i,j],\{k,l,d\}] - 2*\text{sum}[t[l,a]*t[j,k,b,c]*v \\ & + \text{sum}[t[i,c]*t[l,a]*t[j,k],\{k,l,c\}] - 2*\text{sum}[t[l,c]*t[i,k,a,b]*v[k,l,c,d],\{k,l,c,d\}] \end{aligned}$$

Some additional information about the CCSD method...

- $f[i,a]$  and  $t[i,a]$  are rank-2 tensors
- $v[i,j,a,b]$  and  $t[i,j,a,b]$  are rank-4 tensors
- $f$ ,  $v$ , and  $t$  have permutational symmetry properties in their indices, e.g.,  $t[i,j,a,b] = -t[j,i,a,b] = -t[i,j,b,a] = t[j,i,b,a]$
- $f$ ,  $v$ , and  $t$  are block sparse in patterns dictated by molecular symmetries (and permutational symmetries)
- Indices  $i,j,k,l$  refer to “occupied orbitals”
- Indices  $a,b,c,d$  refer to “virtual orbitals”

Theory	# Ter		
CCD	11		
CCSD	48	13,213	1982
CCSDT	102	33,932	1988
CCSDTQ	183	79,901	1992

# Benefits of DSLs for Computers

- Preserve domain-specific information which would be lost in translation to general purpose language
- Use domain-specific information to improve implementation
- Constrained (focused) environment may allow more/better/easier optimizations
- Higher-level specification of computation gives compiler more leeway in translating to target platform

# The Tensor Contraction Engine

## Oak Ridge National Laboratory

**David E. Bernholdt**, Venkatesh  
Choppella, **Robert Harrison**

## University of Florida

**So Hirata**

## Louisiana State University

**Gerald Baumgartner, J  
Ramanujam**



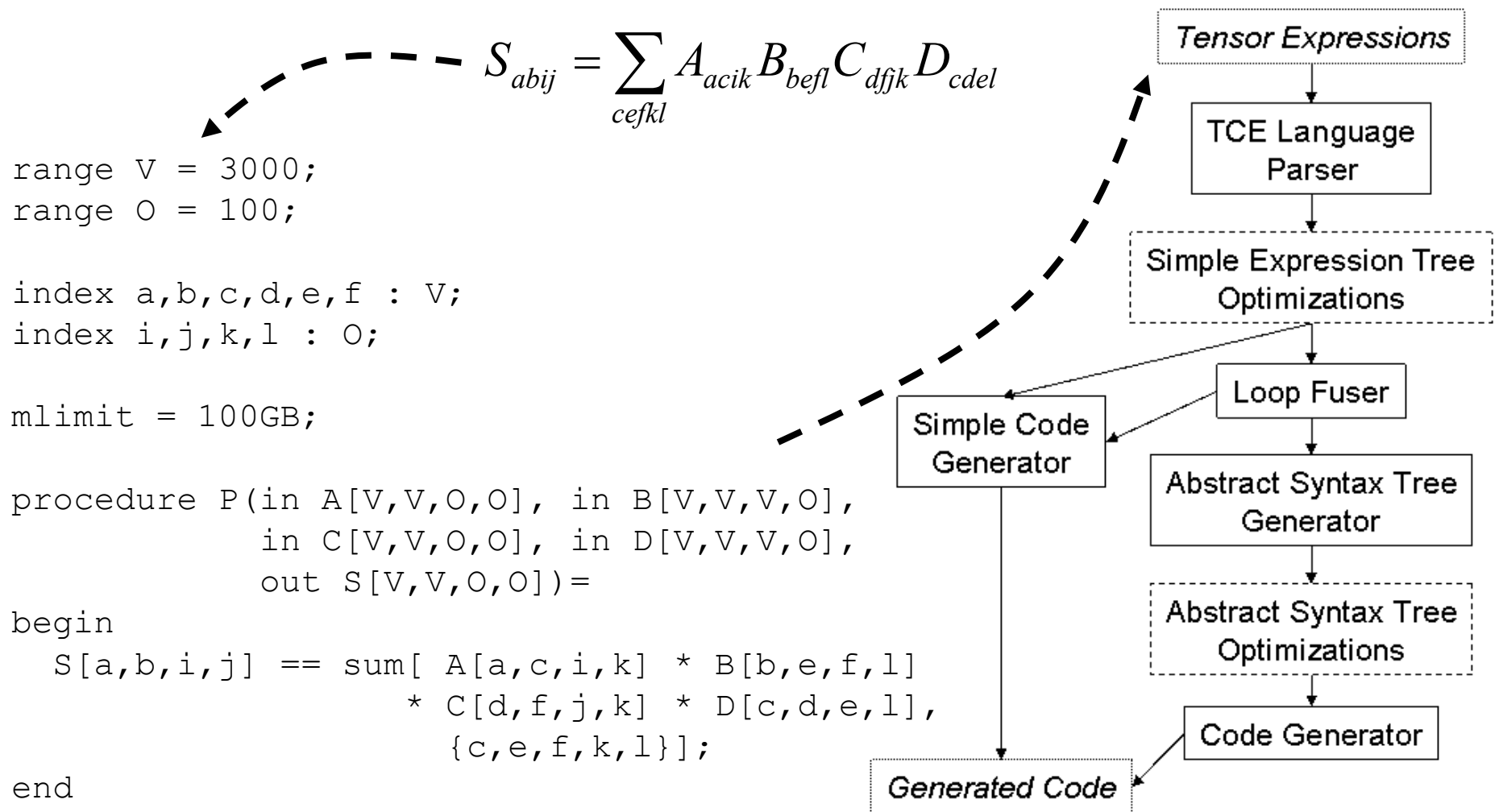
## Ohio State University

Alina Bibireata, Uday  
Bondhugula, Daniel Cociorva,  
Xiaoyang Gao, Albert Hartono,  
Sriram Krishnamoorthy,  
Sandhya Krishnan, Chi-Chung  
Lam, Quingda Lu, **Russell M.  
Pitzer, P Sadayappan**,  
Alexander Sibiryakov

## University of Waterloo

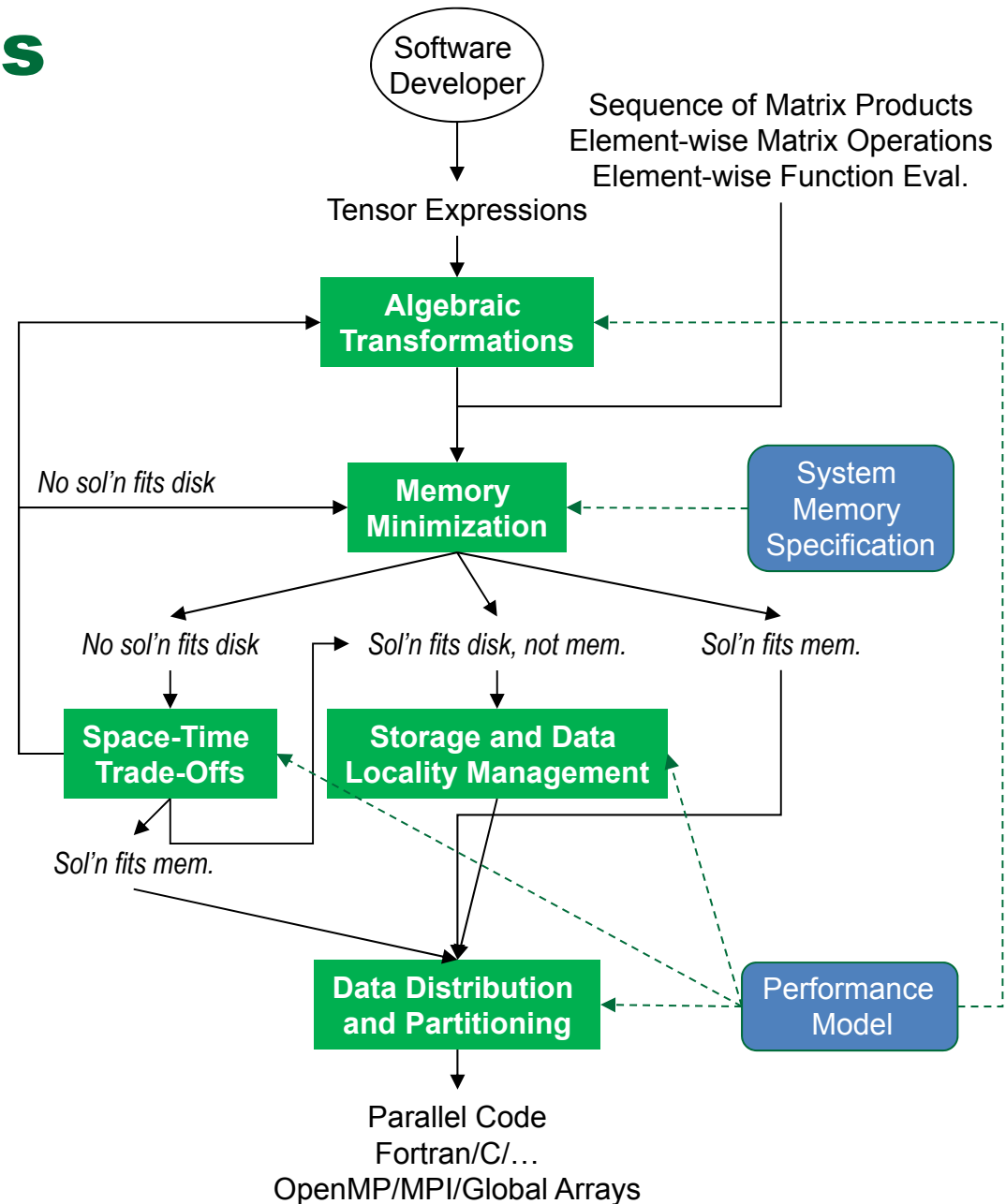
**Marcel Nooijen**, Alexander  
Auer

# TCE Language and Software Architecture



# TCE Optimizations

- Algebraic Transformations
  - Minimize operation count (ICCS'05, ICCS'06)
- Memory Minimization
  - Reduce intermediate storage via loop fusion (LCPC'03)
- Space-Time Transformation
  - Trade-offs between storage and recomputation (PLDI'02)
- Data Locality Optimization
  - Optimize use of storage hierarchy via tiling (ICS'01, HiPC'03, IPDPS'04)
- Data Dist./Comm. Optimization
  - Optimize parallel data layout (IPDPS'03)
- Integrated System
  - (SC'02, Proc. IEEE 05)



# Example: Single Term Optimizations

$$S(a,b,i,j) = \sum_{c,d,e,f,k,l} A(a,c,i,k)B(b,e,f,l)C(d,f,j,k)D(c,d,e,l) \quad 4N^{10} \text{ Ops}$$



$$S(a,b,i,j) = \sum_{c,k} A(a,c,i,k) \left[ \sum_{d,f} C(d,f,j,k) \left( \sum_{e,l} B(b,e,f,l)D(c,d,e,l) \right) \right]$$

---


$$T1(b,c,d,f) = \sum_{e,l} B(b,e,f,l)D(c,d,e,l) \quad 2N^6 \text{ Ops}$$

$$T2(b,c,j,k) = \sum_{d,f} T1(b,c,d,f)C(d,f,j,k) \quad 2N^6 \text{ Ops}$$

$$S(a,b,i,j) = \sum_{c,k} T2(b,c,j,k)A(a,c,i,k) \quad 2N^6 \text{ Ops}$$

# Example: Multi-Term Optimization (Factorization)

- Unoptimized:

$$r_{ij}^{ab} = \sum_{c,d} t_i^c s_j^d v_{cd}^{ab} + \sum_{c,d} u_{ij}^{cd} v_{cd}^{ab} \rightarrow 2O^2V^4 + 3O^2V^4 \text{ ops}$$

- Single-term optimization:

$$r_{ij}^{ab} = \sum_d \left( \sum_c t_i^c v_{cd}^{ab} \right) s_j^d + \sum_{c,d} u_{ij}^{cd} v_{cd}^{ab} \rightarrow 2O^2V^4 + 2OV^4 + 2O^2V^3 \text{ ops}$$

- Factorization:

$$r_{ij}^{ab} = \sum_{c,d} \left( t_i^c s_j^d + u_{ij}^{cd} \right) v_{cd}^{ab} \rightarrow 2O^2V^4 + O^2V^2 \text{ ops}$$

- Improved operation count over single-term optimization

# Lessons Learned from the TCE (1)

- DSLs can have a profound effect on productivity
  - Implementation time of a new coupled cluster method reduced from years to days (hours)
  - Of ~4.5M lines of code in NWChem, approx. 3M+ have been generated by a TCE prototype
- Rich opportunities for optimization
  - Humans have a pretty good intuition for individual optimizations...
  - But not so good with multiple optimizations (combinatorial explosion)
  - Computers are patient and thorough
- Specialized, time-consuming optimizations may be worth the wait
  - If your simulation requires a week or a month on an exascale system, what's the harm in letting the compiler grind away for a few hours to better optimize it?

## Lessons Learned from the TCE (2)

- Important to consider generality of optimizations, tools
  - Easy for everything to end up domain specific
  - Structure of tools can help with generality
  - Requires long, careful discussions with domain experts
- Full language vs code generator to plug into some other framework vs embedding in a general purpose language?
  - TCE code relies on NWChem as part of “runtime”
  - User has to write driver for iteration, convergence
- It is a lot of work to produce a quality “deep” DSL!
  - Designing and implementing core language
  - Optimizations, multiple backends
  - Creating or interfacing with infrastructure

# **Toward a Sustainable Environment for Creating Sustainable DSLs**

- Some aspects of creating DSLs are always going to require work
  - Developing a common understanding between domain scientists and computer scientists
  - Doing a thoughtful analysis of the domain and designing a language for it
- Some aspects we can make less work
  - Developing the general purpose parts of the DSL
  - Targeting different backends/platforms
  - Developing/interfaces with the infrastructure

# Embedded DSLs – Leveraging General Purpose Languages (GPLs)

- The significance of a DSL is the domain-specific part
- But in most cases you need more “around” it
  - Loops, conditionals, basic operations on basic data types, ...
  - Building a *complete* language requires much more work than focusing on a domain-specific core
- Solution: embed DSL in a general purpose language
  - “Small” DSLs only make sense this way
  - Can facilitate interfacing for “large” DSLs
  - Reuse existing language tool chain & environment
  - Possible disadvantage: makes it easier for programmer to go “outside” of DSL

# Which Host Language?

- Rich type system, expressive, extensible
- OO and/or functional features, generic programming
- High performance, sufficiently familiar to programmers
- Exascale features: asynchrony, data distribution, scalable & lightweight synchronization, locality control
- Fortran? C?
- C++?
- PGAS? (Co-Array Fortran, UPC)
- Scala?
- APGAS? (Chapel, X10)

# APGAS Global View Makes for Natural Presentation of Parallel Data Structures

Simple  
TCE input

```
range V = 3000;  
range O = 100;  
  
index a,b,c,d,e,f : V;  
index i,j,k,l : O;  
  
mlimit = 100GB;  
  
procedure P(in A[V,V,O,O], in B[V,V,V,O],  
            in C[V,V,O,O], in D[V,V,V,O],  
            out S[V,V,O,O])=  
  
begin  
    S[a,b,i,j] == sum[ A[a,c,i,k] * B[b,e,f,l]  
                      * C[d,f,j,k] * D[c,d,e,l],  
                      {c,d,e,f,k,l}];  
  
end
```

Chapel version  
by Brad Chamberlain, Cray  
(working code!)

```
config const V = 3000,  
            O = 100;  
const DV = 1..V,  
      DO = 1..O;  
const DVVVOO = [DV, DV, DO, DO],  
      DVVVO = [DV, DV, DV, DO];  
var A, C, S: [DVVVOO] real,  
     B, D: [DVVVO] real;  
forall (a, b, i, j) in DVVVOO do  
    S(a,b,i,j) = + reduce [(c,d,e,f,k,l) in [DV,DV,DV,DV,DO,DO]]  
                      (A(a,c,i,k) * B(b,e,f,l) * C(d,f,j,k) * D(c,d,e,l));  
);
```

# Creating DSLs without Creating New Languages

- Modern languages are increasingly using libraries as an intrinsic part of their design
  - Separate core language elements from “conveniences” that can be built on the core
  - Examples: C `stdlib`; C++ STL, Boost; Java *everything*; ...
- Chapel supports...
  - Generic programming
  - Operator overloading
  - Complex data structures
  - User-defined data distributions
  - User-defined (parallel) iterators
- Do we even need to *extend* the language when we have such features available?

# Turning Libraries into Languages

- Libraries are commonly used to provide domain-specific abstractions without the syntax
- But libraries are black boxes – immutable and opaque
- What if libraries carried with them (machine-actionable) meta-information about their internals, how they could be specialized or transformed?
  - Like Telescoping Languages, but more
  - Use DSLs for compiler transformations
  - Extend X10 to utilize meta-information (written in X10)
- Allow compiler to reason about, optimize library-provided operations
- Makes it easier for DSL developer to leverage libraries into core infrastructure

# Language and Runtime Support for Effective Exascale Execution (LARUS)

## Proposal to 2012 X-Stack Research

- **Oak Ridge National Laboratory**
- IBM
- Ohio State University
- Pacific Northwest National Laboratory
- Rice University
- University of Houston
- University of Illinois
- Cray
- NVIDIA
- APGAS languages as a base
- Base language and DSL-related capabilities
- Compiler optimizations and back-end code generation
- Runtime scalability and adaptivity
- Resilience
- Power and energy
- Tools
- Migration paths

## Summary

- Computational science and engineering applications will constitute a significant part of the cost of exascale computing
- The exascale hardware environment will be notably different than computational scientists have dealt with in the past
- Need to simplify task of mapping equations to code *and* code to hardware
- DSLs are one means to facilitate mapping equations to code
  - Significant benefits, but non-negligible costs
- Appropriate underlying GPL facilitates the second
- Embedding in GPLs simplifies DSL development, leverages existing tools and environment
- Rich GPL may make DSLs unnecessary in some cases
- Annotated libraries to simplify DSL creation